Algoritmos e Estrutura de Dados II

Aula 03

Revisão de Programação Orientada à Objetos

POO

Prof. Dr. Dilermando Piva Jr

2º Semestre - CDN

O que é Orientação a Objetos

Paradigma?

Paradigmas
de Linguagem de Programação

- PARADIGMA?
  - Conjunto de crenças, valores e técnicas compartilhadas por membros de uma determinada comunidade.
  - Conjunto de regras (forma de pensar) utilizadas para resolver problemas!

Paradigmas
de Linguagem de Programação

- PARADIGMA?
  - Conjunto de crenças, valores e técnicas compartilhadas por membros de uma determinada comunidade.
  - Conjunto de regras (forma de pensar) utilizadas para resolver problemas!

- PARADIGMAS DE PROGRAMAÇÃO...
  - Estilo de programação. Uma metodologia utilizada para resolver um problema de forma computacional.

Paradigmas
de Linguagem de Programação

- QUAIS SÃO OS PRINCIPAIS PARADIGMAS DE PROGRAMAÇÃO?
  - Imperativo / Estruturado / Procedural
  - Orientado a Objetos
  - Orientado a eventos
  - Funcional
  - Declarativo
  - Lógico

Paradigmas
de Linguagem de Programação

- QUAIS SÃO OS PRINCIPAIS PARADIGMAS DE PROGRAMAÇÃO?
  - Imperativo / Estruturado / Procedural
    - Foco: em como deve ser feito. Atrelado a FUNÇÃO.
    - Cobol, Pascal, C, Python etc.
  - Orientado a Objetos
  - Orientado a eventos
  - Funcional
  - Declarativo
  - Lógico

Paradigmas
de Linguagem de Programação

- QUAIS SÃO OS PRINCIPAIS PARADIGMAS DE PROGRAMAÇÃO?
  - Imperativo / Estruturado / Procedural
  - Orientado a Objetos
    - Foco está nos elementos componentes da solução (objetos)
    - Cada elementos possui dados e comportamentos
    - Java, C++, PHP, Ruby, Python
  - Orientado a eventos
  - Funcional
  - Declarativo
  - Lógico

Paradigmas
de Linguagem de Programação

- QUAIS SÃO OS PRINCIPAIS PARADIGMAS DE PROGRAMAÇÃO?
  - Imperativo / Estruturado / Procedural
  - Orientado a Objetos
  - Orientado a eventos
    - O foco está no tratamento de eventos (acontecimentos), como um clicar do botão do mouse ou o recebimento de um email.
    - Visual Basic, Delphi, C#
  - Funcional
  - Declarativo
  - Lógico

Paradigmas
de Linguagem de Programação

- QUAIS SÃO OS PRINCIPAIS PARADIGMAS DE PROGRAMAÇÃO?
  - Imperativo / Estruturado / Procedural
  - Orientado a Objetos
  - Orientado a eventos
  - Funcional
    - Focado em funções... Onde o problema pode ser dividido em blocos e, para sua resolução, são implementadas atribuições que definem variáveis em seu escopo que podem ou não retornar resultados.
    - Haskell, Scheme, LISP, Python
  - Declarativo
  - Lógico

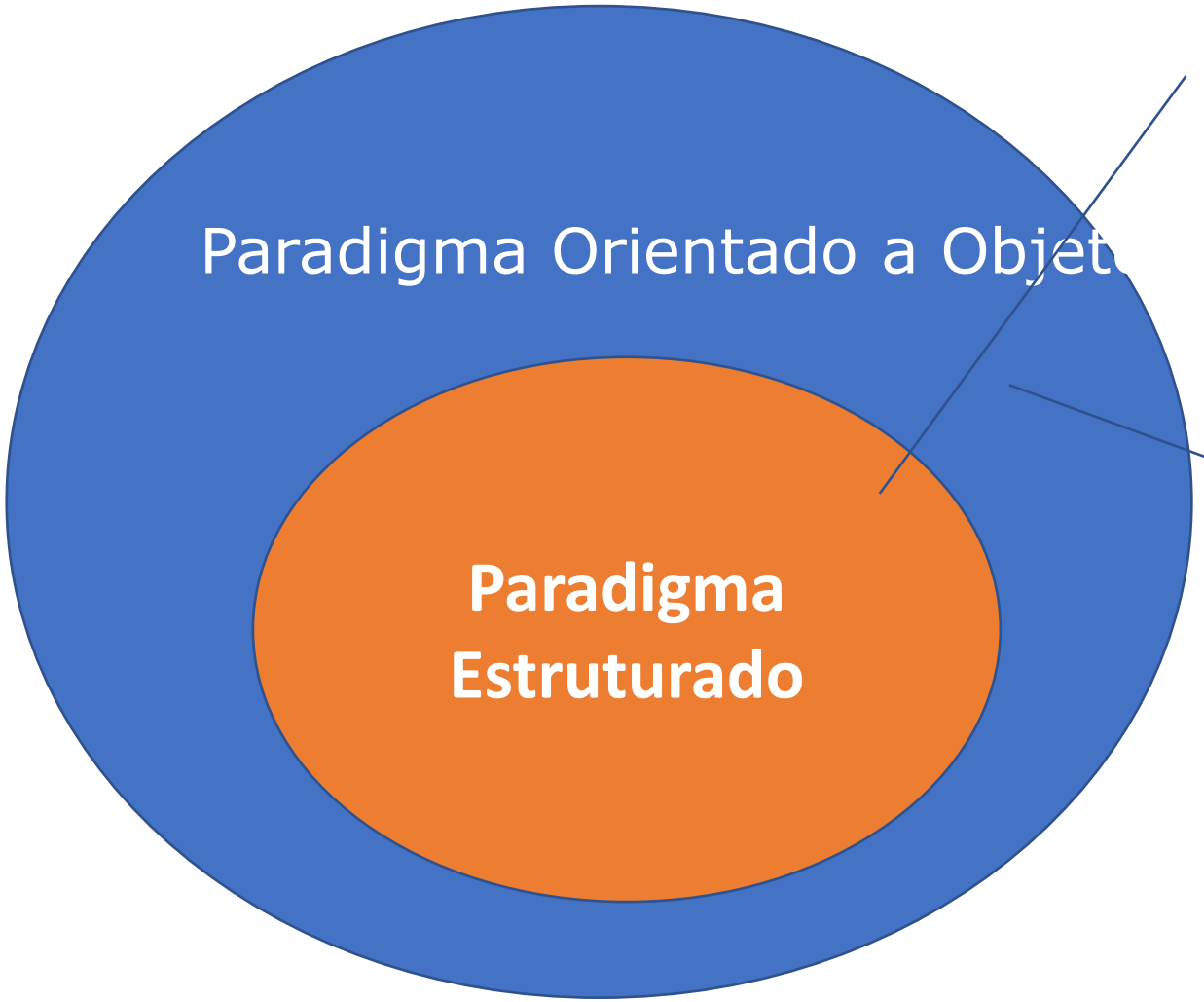
Paradigmas
de Linguagem de Programação

- QUAIS SÃO OS PRINCIPAIS PARADIGMAS DE PROGRAMAÇÃO?
  - Imperativo / Estruturado / Procedural
  - Orientado a Objetos
  - Orientado a eventos
  - Funcional
  - Declarativo
    - Foco está “no que” deve ser resolvido (ao invés de como)
    - Declaração de verdades lógicas imutáveis
    - HTML, XML, CSS etc.
  - Lógico

Paradigmas
de Linguagem de Programação

- QUAIS SÃO OS PRINCIPAIS PARADIGMAS DE PROGRAMAÇÃO?
  - Imperativo / Estruturado / Procedural
  - Orientado a Objetos
  - Orientado a eventos
  - Funcional
  - Declarativo
  - Lógico
    - Também conhecido como restritivo. Deriva do declarativo.
    - Usa lógica simbólica como padrão de entrada e saída.
    - QLISP, Mercury, Prolog

OO x Estruturado/Imperativo



Sequência
Decisões
Repetições...

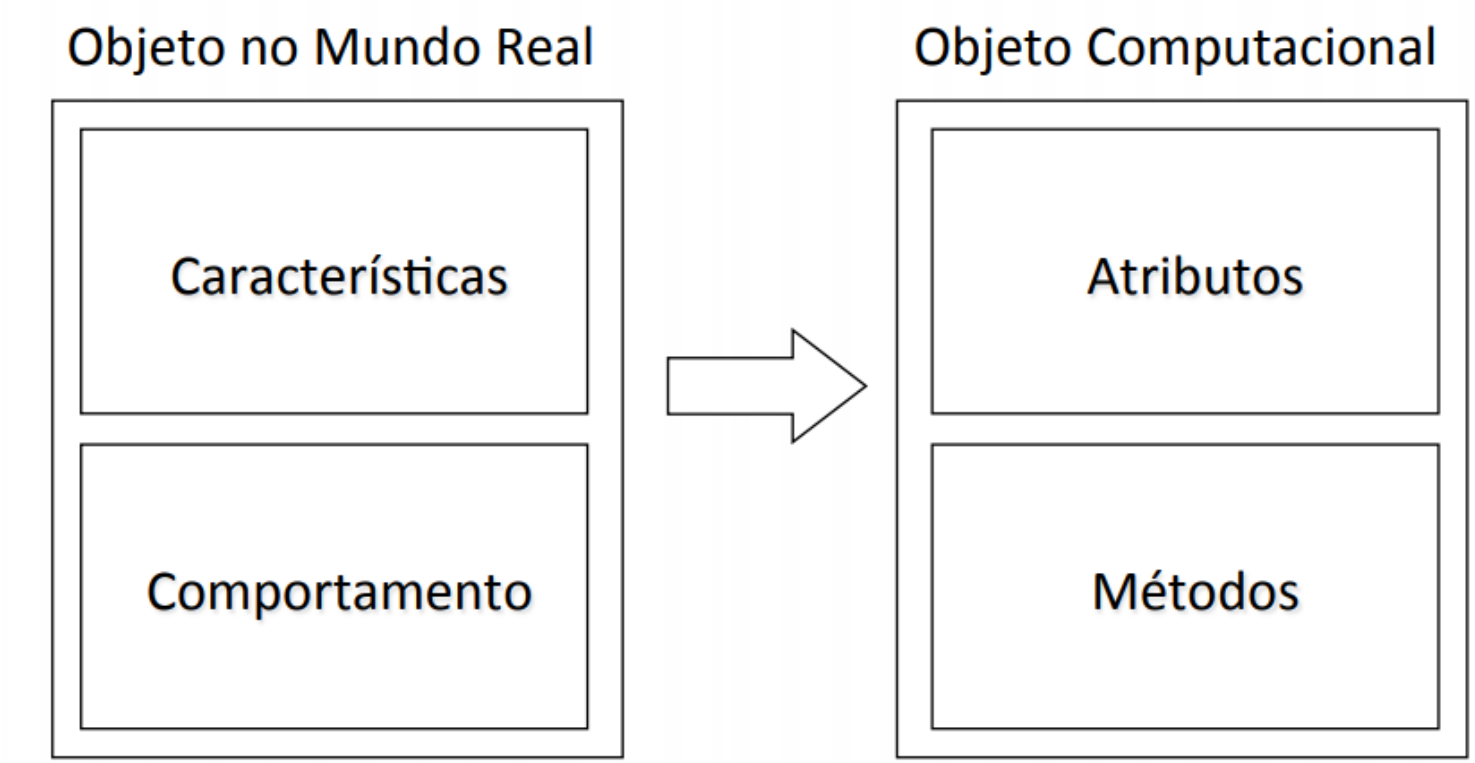
len(nome)

Encapsulamento
Herança
Polimorfismo

nome.len

No mundo real...

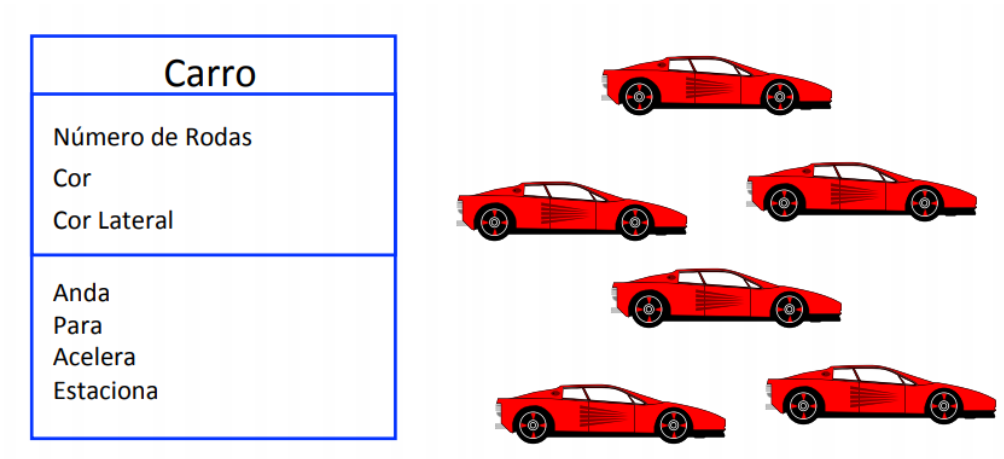
OS ELEMENTOS DO MUNDO REAL... (OBJETOS)



No mundo real...

Características comuns permitem o agrupamento...

Uma Classe...



Paradigma Orientação a Objetos...

- É um paradigma de programação de computadores (conjunto de regras, formas de pensar utilizadas para resolver problemas computacionais)
- O foco está em interpretar a realidade como um conjunto de objetos.
- Os objetos possuem comportamentos (métodos) e características (atributos). Eles são encapsulados e todo o processo de comunicação entre objetos e o mundo real acontece por troca de mensagens.

Paradigma Orientação a Objetos...

Entrevista com **Steve Jobs** *(CEO da Apple)* na Revista Rolling Stone

Jeff Goodell: Você explicaria, em termos simples, exatamente o que é software orientado a objetos?

Paradigma Orientação a Objetos...

Steve Jobs: **Objetos são como pessoas**. Eles estão vivendo, respirando, têm dentro de si conhecimento sobre como fazer as coisas e têm memória dentro deles para que possam se lembrar das coisas.

E em vez de interagir com elas (as pessoas) em um nível muito baixo, você interage com elas em um nível alto de abstração, como estamos fazendo aqui.

Aqui está um exemplo: se eu sou seu **objeto de lavanderia**, você pode me dar suas roupas sujas e me enviar uma **mensagem** dizendo: “Você pode lavar minhas roupas, por favor”.

Acontece que eu sei onde fica a melhor lavanderia de San Francisco. E eu falo inglês e tenho dólares no bolso. Então, eu saio e chamo um táxi e digo ao motorista para me levar a este lugar em São Francisco. Vou lavar suas roupas, volto para o táxi, volto aqui. Eu lhe dou suas roupas limpas e digo: “Aqui estão suas roupas limpas”.

Você não tem ideia de como eu fiz isso. Você não tem conhecimento da lavanderia. Talvez você fale francês e não consiga nem chamar um táxi. Você não pode pagar por um, você

não tem dólares no bolso. No entanto, eu sabia como fazer tudo isso. **E você**

não precisou saber nada disso. **Toda essa complexidade estava escondida**

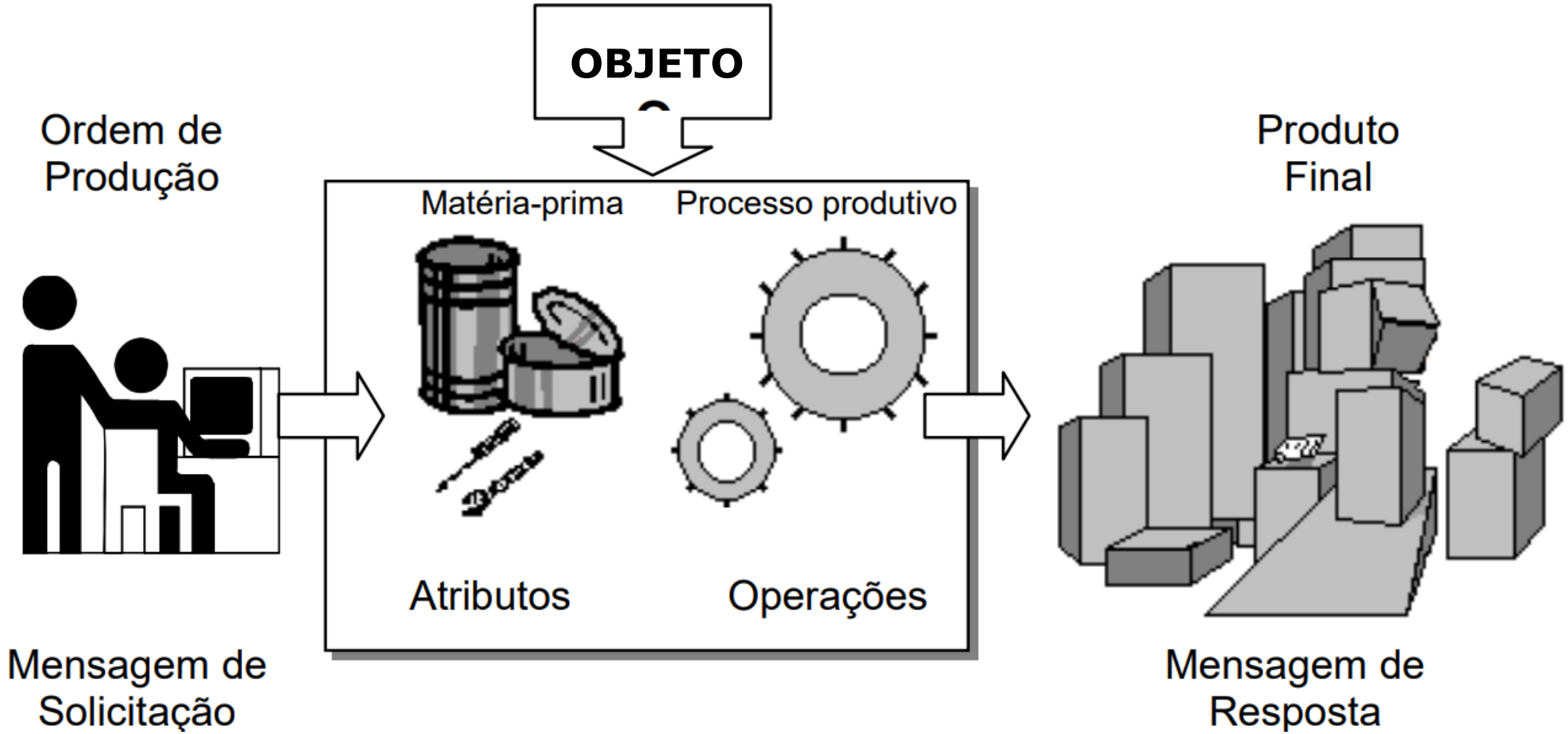
dentro de mim, e pudemos interagir em um nível muito alto de abstração.

Isso é o que são os objetos. Eles encapsulam a complexidade e

disponibilizam as interfaces (de alto nível) para tratar a complexidade.

[Fonte: https://www.rollingstone.com/culture/culture-news/steve-jobs-in-1994-the-rolling-stone-interview-231132/]

Paradigma Orientação a Objetos...



Princípios do Paradigma Orientado a Objetos

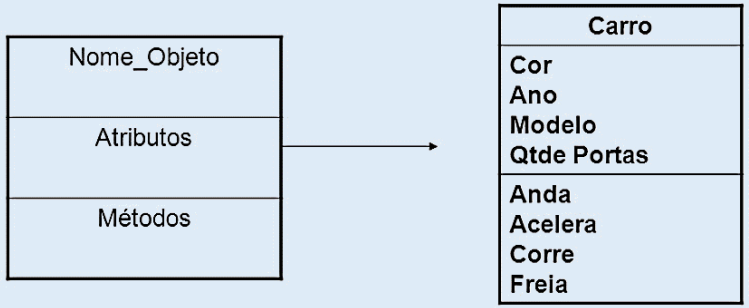
Pilares...

Pilares da O. O.

Pilares da O. O.

ENCAPSULAMENTO

Propriedade de você utilizar um objeto sem entender o seu funcionamento interno.

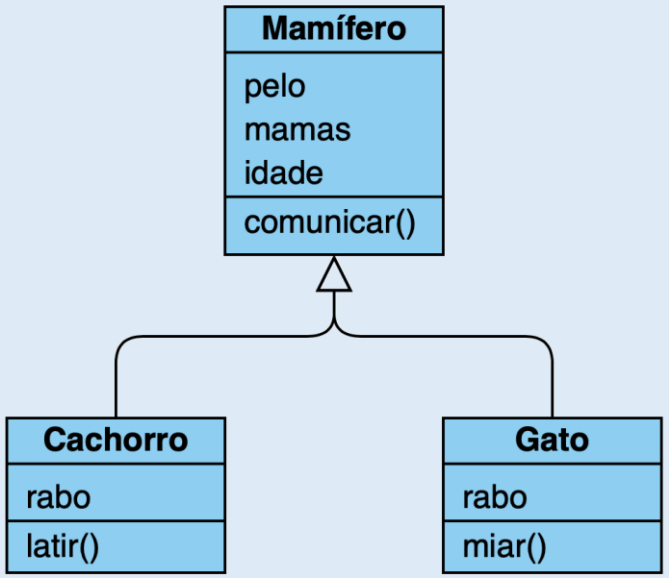


- ENCAPSULAMENTO
- HERANÇA
- POLIMORFISMO
- ABSTRAÇÃO

Pilares da O. O.

HERANÇA

Propriedade de uma classe filha (subclasse) herdar componentes da classe pai (superclasse)



É uma estratégia que possibilita o "**reuso de código**"

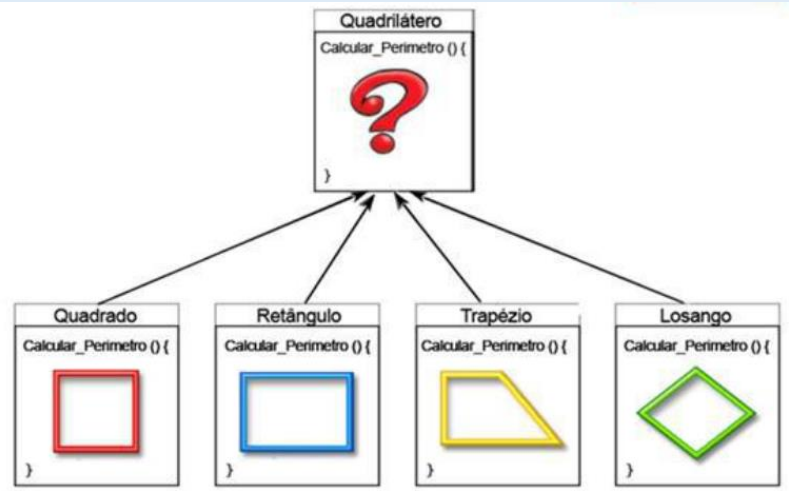
_____ é um(a) _____ ➔ Cachorro **é um** Mamífero.

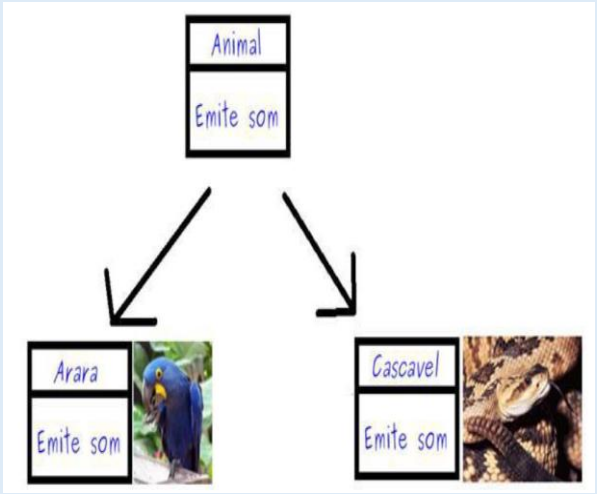
- ENCAPSULAMENTO
- HERANÇA
- POLIMORFISMO
- ABSTRAÇÃO

Pilares da O. O.

POLIMORFISMO

Propriedade que permite que uma mesma mensagem seja enviada a diferentes objetos e que cada objeto execute a operação que é apropriada a sua classe.





sobreposição de métodos.

- ENCAPSULAMENTO
- HERANÇA
- POLIMORFISMO
- ABSTRAÇÃO

Pilares da O. O.

ABSTRAÇÃO

Propriedade de esconder os detalhes da implementação dentro de algo.
Forma utilizada para lidar com a complexidade do mundo real.

Por exemplo: quando chamamos uma função não precisamos saber como ela realizará o processamento interno. Nos interessa apenas o resultado.
Outro exemplo: Quando ligamos o carro, não temos que saber como ele executará todas as tarefas necessárias para que o motor entre em funcionamento.

- ENCAPSULAMENTO
- HERANÇA
- POLIMORFISMO
- ABSTRAÇÃO

Pilares da O. O.

ABSTRAÇÃO

Propriedade de esconder os detalhes da implementação dentro de algo.
Forma utilizada para lidar com a complexidade do mundo real.

Por exemplo: quando chamamos uma função não precisamos saber como ela realizará o processamento interno. Nos interessa apenas o resultado.
Outro exemplo: Quando ligamos o carro, não temos que saber como ele executará todas as tarefas necessárias para que o motor entre em funcionamento.

CLIENTE	CLIENTE
Nome Tipo Sanguíneo Plano de Saúde	Nome Renda Familiar Cartão de Crédito
FazerExame()	Comprar()

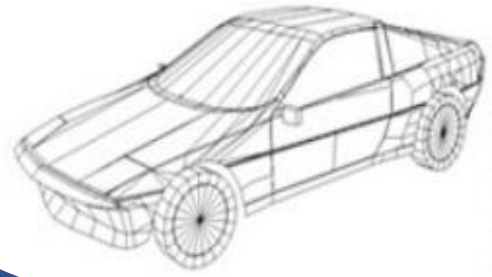
- ENCAPSULAMENTO
- HERANÇA
- POLIMORFISMO
- ABSTRAÇÃO

Relação entre Classe e Objeto

Classe vs Objeto

No mundo real...

CLASSE →



Tipo: ?

Cor: ?

Placa: ?

Número de Portas: ?

Molde

← OBJETOS →

Tipo: Porsche

Cor: Cinza

Placa: MHZ-4345

Número de Portas: 2

Tipo: Ferrari

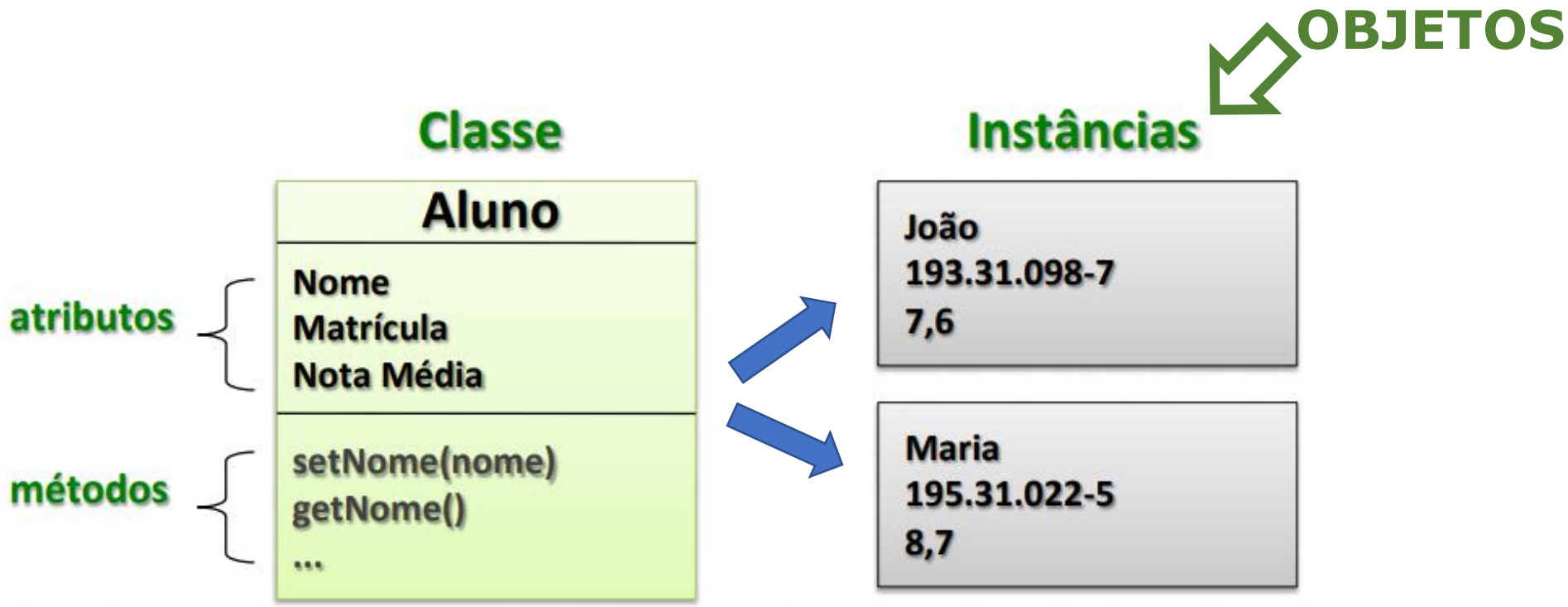
Cor: Vermelho

Placa: JKL-0001

Número de Portas: 4

Instâncias

Computacionalmente...



Criando uma Classe ...

E instanciando seus objetos...

Definindo uma classe...

Recapitulando:
CLASSE....: Molde
OBJETO...: Agente ativo na programação
MÉTODO.: Capacidade de ação do agente ativo
ATRIBUTO: Característica do agente ativo

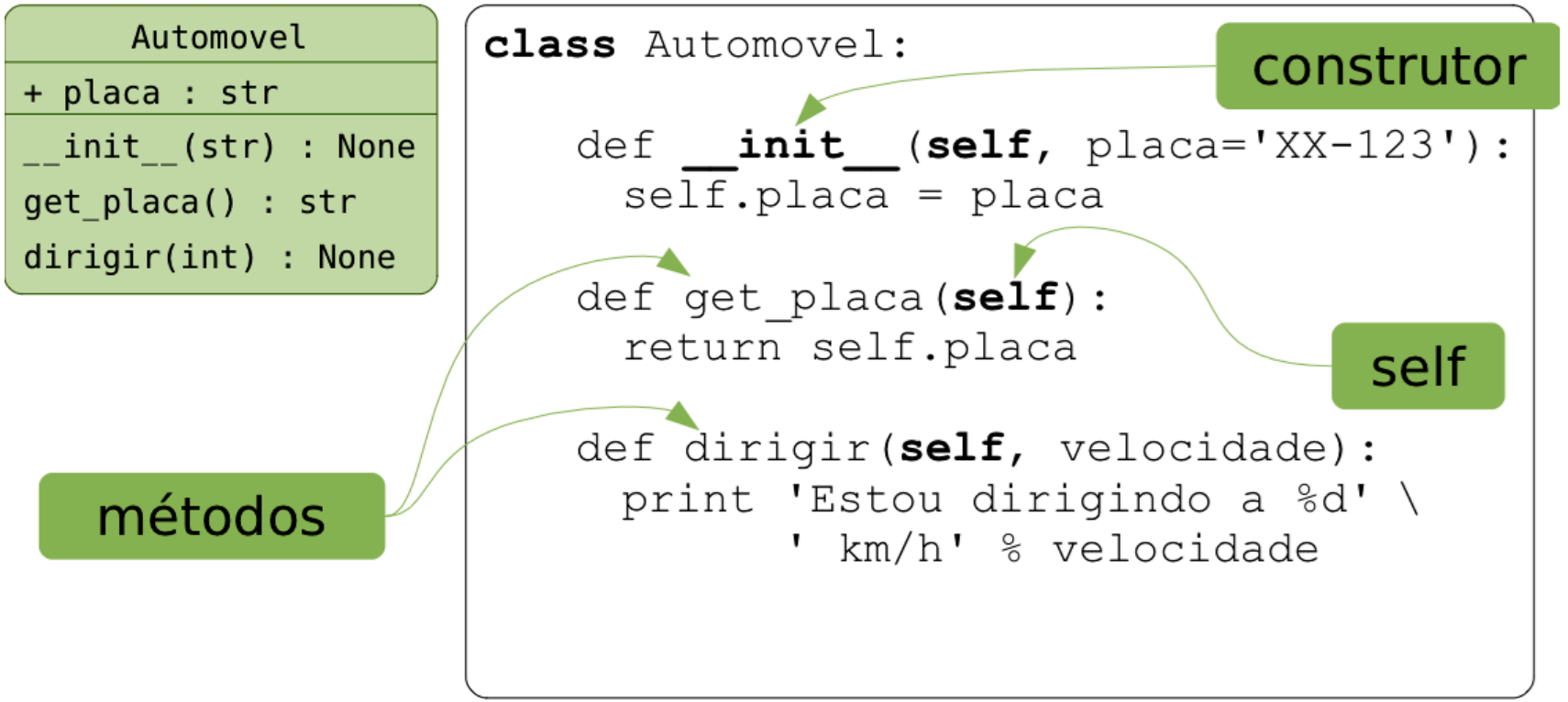
Exemplo:
CLASSE....: Automóvel
OBJETO...: Meu Carro
MÉTODO.: Acelerar(), Frear()...
ATRIBUTO: Placa, Cor, Ano Fabricação...

Python não usa o conceito de definição de interfaces como em outras linguagens. Basta você definir a classe e utilizá-la!

Definindo uma classe...

```
class Automovel:
    pass

meu_carro = Automovel()
```

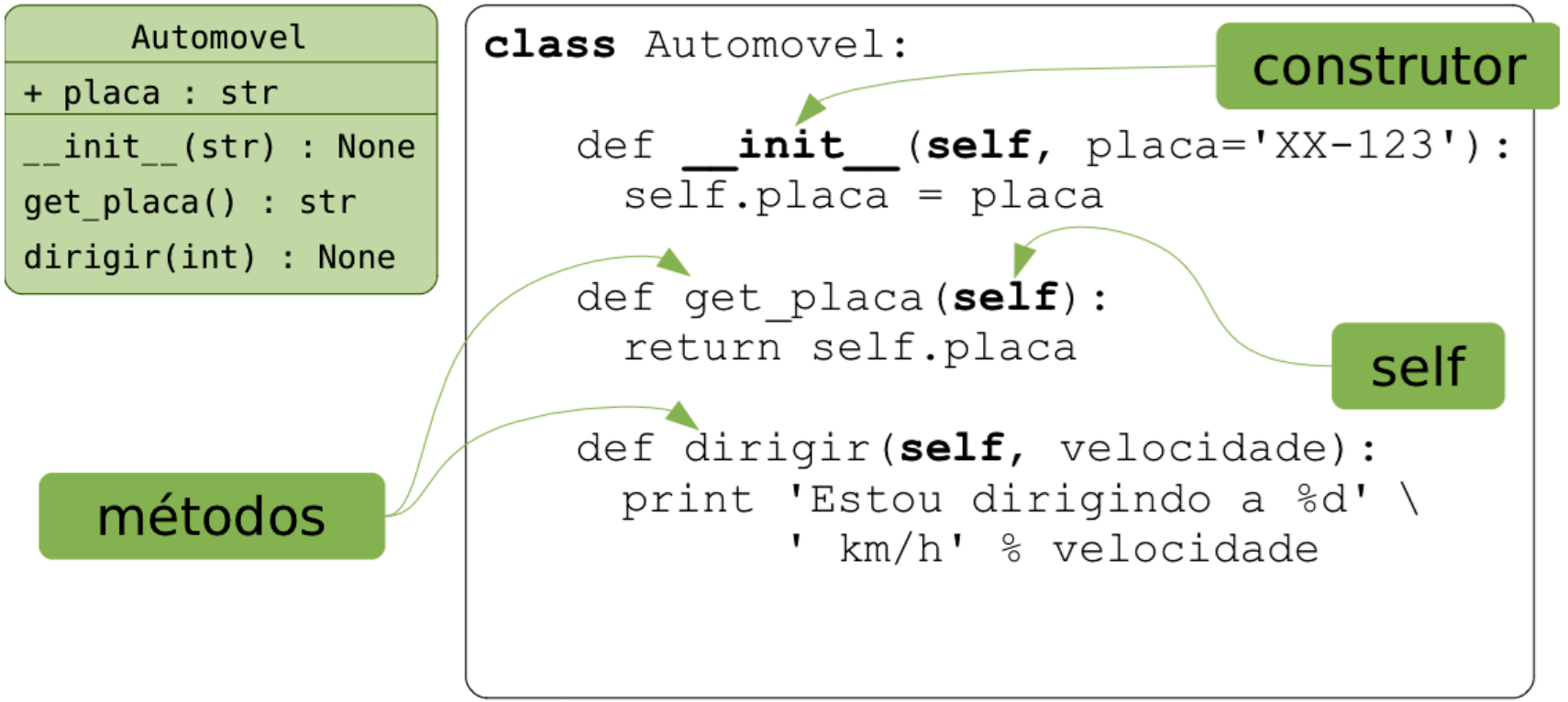
Definição de uma classe



Definindo uma classe...

```
class Automovel:
    def __init__(self, placa):
        self.placa = placa

meu_carro = Automovel('XZX1234')

print(meu_carro.placa)
```

Definição de uma classe



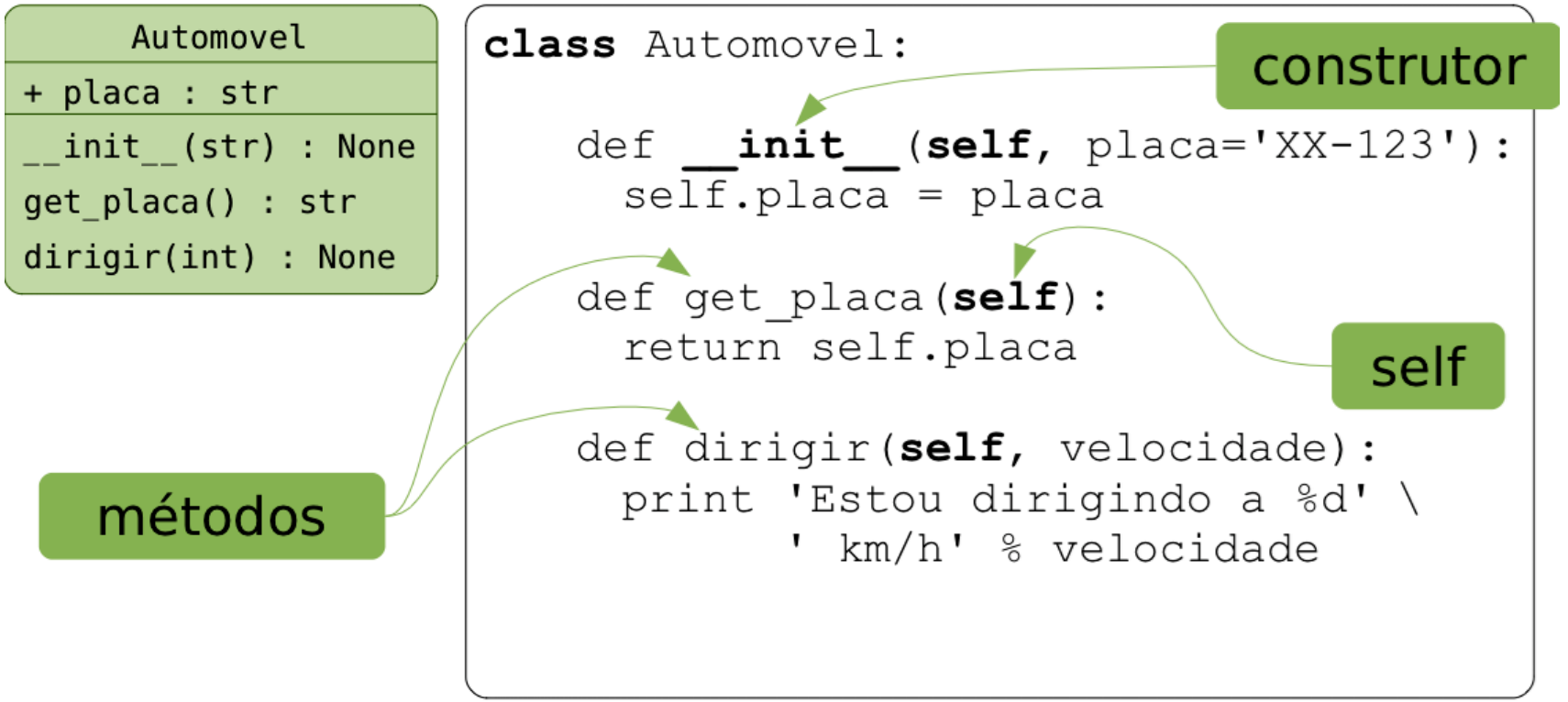
Definindo uma classe...

```
class Automovel:
    def __init__(self, placa):
        self.placa = placa

    def get_placa(self):
        return self.placa

meu_carro = Automovel('XZX1234')

print(meu_carro.get_placa())
```

Definindo uma classe...

```
class Automovel:
    def __init__(self, placa):
        self.placa = placa

    def get_placa(self):
        return self.placa

meu_carro = Automovel('XZX1234')

print(meu_carro.get_placa())

meu_carro.placa = 'ZZZ0011'

print(meu_carro.get_placa())
```

Definição de uma classe



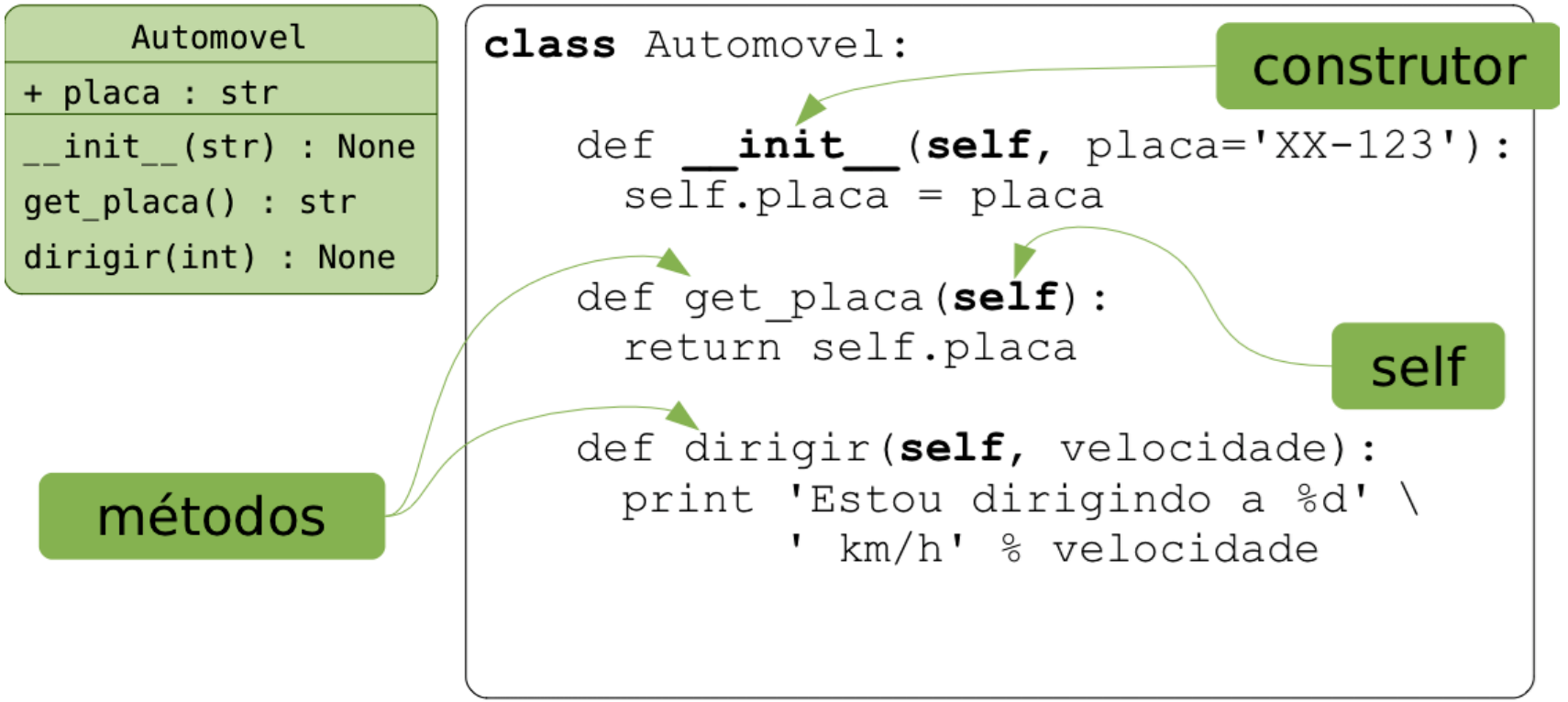
Definindo uma classe...

```
class Automovel:
    def __init__(self, placa):
        self.placa = placa

    def get_placa(self):
        return self.placa

    def dirigir(self, velocidade):
        print(f'Estou digirindo a {velocidade} km/h')

meu_carro = Automovel('XZX1234')

print(meu_carro.get_placa())

meu_carro.digirir(100)
```

Criando mais uma Classe ...

E instanciando seus objetos...

Definição de uma classe



Definindo uma classe...

```
class Automovel:
    def __init__(self, placa):
        self.placa = placa

    def get_placa(self):
        return self.placa

    def dirigir(self, velocidade):
        print(f'Estou digirindo a {velocidade} km/h')

meu_carro = Automovel('XZX1234')

print(meu_carro.get_placa())

meu_carro.digirir(100)
```

Definindo uma classe...

```
class Automovel:
    def __init__(self, placa, velocidade_max):
        self.placa = placa
        self.velocidade_max = velocidade_max

    def to_str(self):
        return f'{self.velocidade_max} Km/h'

    def get_placa(self):
        return self.placa

    def dirigir(self, velocidade):
        print(f'Estou digirindo a {velocidade} km/h')

meu_carro = Automovel('XZX1234')
print(meu_carro.get_placa())
meu_carro.digirir(100)
print(meu_carro.to_str())
```

Definindo uma classe...

```
class Automovel:
    def __init__(self, placa, velocidade_max):
        self.placa = placa
        self.velocidade_max = velocidade_max

    def to_str(self):
        return f'{self.velocidade_max} Km/h'

    def __str__(self):
        return f'{self.velocidade_max} Km/h'

    def get_placa(self):
        return self.placa

    def dirigir(self, velocidade):
        print(f'Estou digirindo a {velocidade} km/h')

meu_carro = Automovel('XZX1234', 180)
print(meu_carro.get_placa())
meu_carro.digirir(100)
print(meu_carro)
```

Método mágico __str__

É suportado por todos os objetos em Python

Automóvel
str: placa int: velocidade_max
__init__(str) __str__() get_placa() dirigir(velocidade)

Automóvel
str: placa int: velocidade_max
__init__(str) __str__() get_placa() dirigir(velocidade)

Automóvel
str: placa int: velocidade_max int: velocidade_atual
__init__(str) __str__() get_placa() acelerar() frear()

Acelerar → acrescenta 10 Km até a velocidade_max
Frear → decrementa 10 Km até 0Km

Definindo uma classe...

```
class Automovel:
    def __init__(self, placa, velocidade_max):
        self.placa = placa
        self.velocidade_max = velocidade_max
        self.velocidade_atual = 0

    def __str__(self):
        return f'{self.velocidade_atual} Km/h'

    def get_placa(self):
        return self.placa

    def acelerar(self):

    def frear(self):

meu_carro = Automovel('XZX1234', 180)
```

Definindo uma classe...

```
class Automovel:
    def __init__(self, placa, velocidade_max):
        self.placa = placa
        self.velocidade_max = velocidade_max
        self.velocidade_atual = 0

    def __str__(self):
        return f'{self.velocidade_atual} Km/h'

    def get_placa(self):
        return self.placa

    def acelerar(self):
        maxima = self.velocidade_max
        nova = self.velocidade_atual + 10
        self.velocidade_atual = nova if nova <= maxima else maxima

    def frear(self):
        nova = self.velocidade_atual - 10
        self.velocidade_atual = nova if nova >= 0 else 0
```

```
meu_carro = Automovel('XZX1234', 180

for _ in range(20):
    meu_carro.acelerar()
    print(meu_carro)

for _ in range(20):
    meu_carro.frear()
    print(meu_carro)
```

Atributos Privados ...

Encapsulando os atributos...

Acessando os atributos...

```
class Automovel:
    def __init__(self, placa, velocidade_max):
        self.placa = placa
        self.velocidade_max = velocidade_max
        self.velocidade_atual = 0

    def __str__(self):
        return f'{self.velocidade_atual} Km/h'

    def get_placa(self):
        return self.placa

    def acelerar(self):
        maxima = self.velocidade_max
        nova = self.velocidade_atual + 10
        self.velocidade_atual = nova if nova <= maxima else maxima

    def frear(self):
        nova = self.velocidade_atual - 10
        self.velocidade_atual = nova if nova >= 0 else 0
```

```
meu_carro = Automovel('XZX1234', 180)

print(meu_carro)
print(meu_carro.velocidade_max)
print(meu_carro.get_placa())

#Sem encapsulamento adequado...
meu_carro.placa = 'XXX0000'
meu_carro.velocidade_max = 200
meu_carro.velocidade_atual = 190

print(meu_carro)
print(meu_carro.velocidade_max)
print(meu_carro.get_placa())
```

Acessando os atributos...

```
class Automovel:
    def __init__(self, placa, velocidade_max):
        self.placa = placa
        self.velocidade_max = velocidade_max
        self.velocidade_atual = 0

    def __str__(self):
        return f'{self.velocidade_atual} km/h'

    def get_placa(self):
        return self.placa

    def acelerar(self):
        maxima = self.velocidade_max
        nova = self.velocidade_atual + 10
        self.velocidade_atual = nova if nova <= maxima else maxima

    def frear(self):
        nova = self.velocidade_atual - 10
        self.velocidade_atual = nova if nova >= 0 else 0
```

Viola o princípio do encapsulamento dos dados

```
meu_carro = Automovel('XZX1234', 180)

print(meu_carro)
print(meu_carro.velocidade_max)
print(meu_carro.get_placa())

#Sem encapsulamento adequado...
meu_carro.placa = 'XXX0000'
meu_carro.velocidade_max = 200
meu_carro.velocidade_atual = 190

print(meu_carro)
print(meu_carro.velocidade_max)
print(meu_carro.get_placa())
```

Acessibilidade...

Atributos e Métodos

PRIVADOS (Privated)

Para tornar os métodos e os atributos acessíveis apenas dentro da própria classe, utilizamos “__” antes de seus nomes.

PROTEGIDOS (Protected)

Para tornar os métodos protegidos (apenas uma convenção... Não altera a proteção... Eles continuam sendo acessados de fora) utilizamos “_” antes de seus nomes.

Definindo uma classe...

```
class Automovel:
    def __init__(self, placa, velocidade_max):
        self.__placa = placa
        self.__velocidade_max = velocidade_max
        self.__velocidade_atual = 0

    def __str__(self):
        return f'{self.__velocidade_atual} Km/h'

    def get_placa(self):
        return self.__placa

    def acelerar(self):
        maxima = self.__velocidade_max
        nova = self.__velocidade_atual + 10
        self.__velocidade_atual = nova if nova <= maxima else maxima

    def frear(self):
        nova = self.__velocidade_atual - 10
        self.__velocidade_atual = nova if nova >= 0 else 0
```

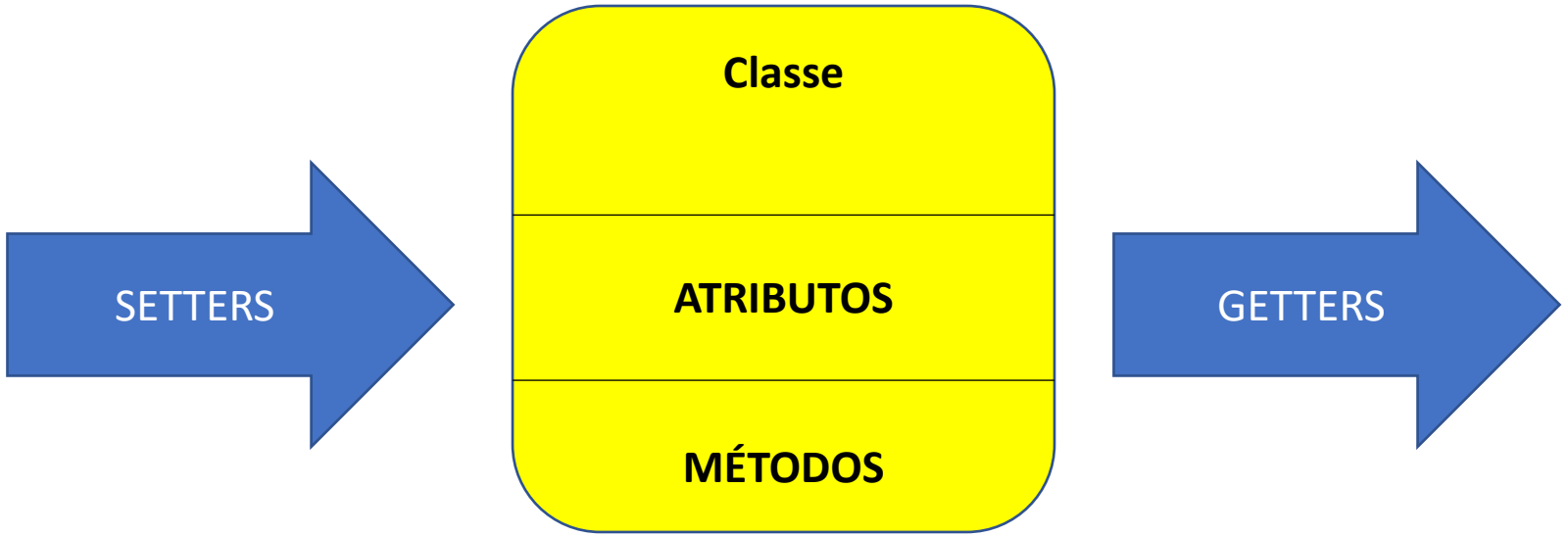
```
meu_carro = Automovel('XZX1234', 180)

print(meu_carro)
print(meu_carro.__velocidade_max)
print(meu_carro.get_placa())

#Sem encapsulamento adequado...
meu_carro.__placa = 'XXX0000'
meu_carro.__velocidade_max = 200
meu_carro.__velocidade_atual = 190

print(meu_carro)
print(meu_carro.__velocidade_max)
print(meu_carro.get_placa())
```

Métodos getters e setters...

Trocando mensagens com os objetos!

Definindo a interface...

Definindo a interface...



Acessando atributos via interface

```
class Automovel:
    def __init__(self, placa, velocidade_max):
        self.__placa = placa
        self.__velocidade_max = velocidade_max
        self.__velocidade_atual = 0

    def __str__(self):
        return f'{self.__velocidade_atual} Km/h'

    def get_placa(self):
        return self.__placa

    def get_velocidade_max(self):
        return self.__velocidade_max

    def set_velocidade_max(self, nova):
        self.__velocidade_max = nova

    def acelerar(self):
        maxima = self.velocidade_max
        nova = self.velocidade_atual + 10
        self.velocidade_atual = nova if nova <= maxima else maxima

    def frear(self):
        nova = self.velocidade_atual - 10
        self.velocidade_atual = nova if nova >= 0 else 0
```

```
meu_carro = Automovel('XZX1234', 180)

print(meu_carro)
print(meu_carro.get_placa())
print(meu_carro.get_velocidade_max())

#Com encapsulamento adequado...
meu_carro.set_velocidade_max(200)

print(meu_carro)
print(meu_carro.get_placa())
print(meu_carro.get_velocidade_max())
```

Acessando atributos via interface

```
class Automovel:
    def __init__(self, placa, velocidade_max):
        self.__placa = placa
        self.__velocidade_max = velocidade_max
        self.__velocidade_atual = 0

    def __str__(self):
        return f'{self.__velocidade_atual} Km/h'

    def get_placa(self):
        return self.__placa

    def get_velocidade_max(self):
        return self.__velocidade_max

    def set_velocidade_max(self, nova):
        self.__velocidade_max = nova

    def acelerar(self):
        maxima = self.velocidade_max
        nova = self.velocidade_atual + 10
        self.velocidade_atual = nova if nova <= maxima else maxima

    def frear(self):
        nova = self.velocidade_atual - 10
        self.velocidade_atual = nova if nova >= 0 else 0
```

```
meu_carro = Automovel('XZX1234', 180)

print(meu_carro)
print(meu_carro.get_placa())
print(meu_carro.get_velocidade_max())

#Com encapsulamento adequado...
meu_carro.set_velocidade_max(200)

print(meu_carro)
print(meu_carro.get_placa())
print(meu_carro.get_velocidade_max())
```

Atributos de Classe...

Em Python... Atributos estáticos!

Atributos de Classe...

```
class Automovel:
    contador = 0
    precisao = 0.95  # 5%
    def __init__(self, placa, velocidade_max):
        self.__id = Automovel.contador + 1
        self.__placa = placa
        self.__velocidade_max = velocidade_max * Automovel.precisao
        self.__velocidade_atual = 0
        Automovel.contador = self.__id

    def __str__(self):
        return f'{self.__id} - {self.__velocidade_atual} Km/h'

    def get_placa(self):
        return self.__placa

    def get_velocidade_max(self):
        return self.__velocidade_max

    def set_velocidade_max(self, nova):
        self.__velocidade_max = nova

    def acelerar(self):
        maxima = self.velocidade_max
        nova = self.velocidade_atual + 10
        self.velocidade_atual = nova if nova <= maxima else maxima

    def frear(self):
        nova = self.velocidade_atual - 10
        self.velocidade_atual = nova if nova >= 0 else 0
```

```
meu_carro = Automovel('XZX1234', 180)
seu_carro = Automovel('XXX0000', 200)

print(meu_carro)
print(meu_carro.get_placa())
print(meu_carro.get_velocidade_max())

print(seu_carro)
print(seu_carro.get_placa())
print(seu_carro.get_velocidade_max())
```

VAMOS PARA A PRÁTICA ?!!!

Classe Usuário (com senha!)

- Crie uma classe Usuario com os seguintes atributos: nome, sobrenome, email e senha.
- Todos os atributos devem ser privados.
- Crie dois métodos:
  - nome_completo (que irá imprimir o nome completo – Nome + Sobrenome.
  - checa_senha (que receberá como parâmetro uma str contendo uma senha e ela será comparada a que foi utilizada para criar o usuário. Se Igual, retorna True... Caso contrário, False.
- Para guardar a senha... Vamos utilizar um pacote de criptografia de senha chamado passlib (Terceiros)

Classe Usuário (com senha!)

- Instalar o Módulo: pip install passlib
- Vamos utilizar um algoritmo específico de criptografia sha256. Sugestão de importação:
  - from passlib.hash import pbkdf2_sha256 as cryp
- Vamos utilizar dois métodos específicos:
  - encrypt (<senha>, rounds=n, salt_size=x)
  - Exemplo: crip.encrypt(senha, rounds=1000, salt_size=10)

  - verify(<senha>, <senha_guardada>)
  - Exemplo: crip.verify(senha, self.__senha)

Classe do usuário com senha - resposta

```
from passlib.hash import pbkdf2_sha256 as cryp

class Usuario:
         def __init__(self, nome, sobrenome, email, senha):
        self.__nome = nome
        self.__sobrenome = sobrenome
        self.__email = email
        self.__senha = cryp.encrypt(senha, rounds=1000, salt_size=10)

    def nome_completo(self):
        return f'{self.__nome} {self.__sobrenome}'

    def checa_senha(self, senha):
        if cryp.verify(senha, self.__senha):
             return True
        return False
```

Classe do usuário com senha - resposta

```
while True:
    nome = input(“Nome: “)
    sobrenome = input(“Sobrenome: “)
    email = input(“Email: “)
    senha = input(“Senha: “)
    confirma_senha = input(“Confirme a senha: “)
    if senha == confirma_senha:
        pessoa = Usuario(nome, sobrenome, email, senha)
        break
    else
        print(“Senha não confere...”)

print(“Usuário Criado com Sucesso!!!)

senha = input(“Informe a senha para acesso: “)

if pessoa.checa_senha(senha)
    print(“Acesso permitido!)
else
    print(“Acesso negado!)
```